TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Optimalizace distribuce dat při paralelním řešení úloh proudění a transportu

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Lenka Jandurová**
Vedoucí práce: Mgr. Jan Březina, Ph.D.

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Optimization of data distribution in parallel solution of flow and transport problems

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Lenka Jandurová**
Supervisor: Mgr. Jan Březina, Ph.D.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lenka Jandurová**
Osobní číslo:	**M12000138**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Optimalizace distribuce dat při paralelním řešení úloh proudění a transportu**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Instalujte aktuální verzi programu Flow123d a balíku METIS na paralelním počítači.
2. Připravte testovací úlohy.
3. Otestujte použití různých nástrojů pro sledování výkonu paralelního programu: Intel Trace Analyzer, vlastní profiler projektu, profilovací informace z PETSC.
4. Otestujte kvalitu současného způsobu dělení sítě pro různé numerické metody a lineární řešiče.
5. Implementujte kód pro získání dělení sítě, které zohledňuje dimenze elementů.
6. Otestujte nová dělení a jejich vliv na výkon lineárních řešičů jednotlivých numerických metod.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 60 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. KARYPIS, George.[online]. 2013 [cit. 2014-10-26]. Dostupné z: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

Load Balancing Fictions Falsehoods and Fallacies. HENDRICKSON, Bruce. Sandia National Laboratories: Exceptional Service in the National Interest [online]. 1999 [cit. 2014-10-26]. Dostupné z: http://www.sandia.gov/ bahendr/abstracts/myths.html

Vedoucí bakalářské práce: **Mgr. Jan Březina, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **20. října 2015**

Termín odevzdání bakalářské práce: **16. května 2016**

Technická univerzita v Liberci
L.S.

prof. Ing. Václav Kopecký, CSc.
děkan

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2015

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Mé poděkování patří Mgr. Janu Březinovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval, a také Metacentru, za možnost užívat jejich výpočetní techniku. Ještě bych chtěla poděkovat Ing. Janovi Koprnickému, Ph.D. za vytvoření šablony v sázecím systému Latex.

Abstrakt

Bakalářská práce se zaměřuje kvalitu dělení sítě pro flow123d za použití knihovny BDDCML a METISu. Cílem je toto dělení upravit a zefektivnit tak výpočet. Dále dokázat, že tato změna nebude mít negativní vliv na výpočet pomocí knihovny Petsc. V řešení bylo použito metody vážení, která ovlivňuje váhu vybraných elementů sítě a úprava nastavení METISu. Také byli testovány různé měřící metody.

Abstract

Bachelor thesis focuses on the quality of the network division flow123d using library BDDCML and Metis. The objective this division is to modify and streamline calculation. Additionally prove that this change will not have a negative effect on the calculation using the library PETSc. The solution was using weighting method, which affects the weight of selected network elements and adjusting settings Metis. They were also tested different methods of measurement.

Klíčová slova

Flow123d, Metacentrum, Petsc, BDDC, Metis, Intel trace Analyzer

Keywords

Flow123d, Metacentrum, Petsc, BDDC, Metis, Intel trace Analyzer

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Úvodním motivem této práce je vylepšení stávajícího způsobu dělení sítě při použití programu flow123d. Stávající způsob při použití metody BDDC má problém se zvláštním chováním v některých případech běhu aplikace, jak bylo zjištěno zde: [1]. Jako možná příčina by mohl být způsob dělení sítě při špatném rozdělení prvků s vyšší vodivostí. Proto bude otestována metoda vážení, její vliv na výpočty pod řešičem BDDC a zda nějakým způsobem ovlivní běh pod jiným řešičem, v tomto případě BiCGStab z Petsc.

Program Flow123d je zaměřený na výpočet proudění a transportu v porézních látkách. Pracuje s 1 - 3 rozměrnými elementy a výpočetní síť se skládá z čtyřstěnů, trojúhelníků a úseček. Dále bylo potřeba vhodné prostředí pro běh, které umožňuje paralelní výpočty a má dostatečnou infrastrukturu. Bylo zvoleno Metacentrum, které uvedené požadavky splňuje. Zde bylo nutno vše nakonfigurovat. V této práci bylo použito několik měřících metod a řešena nesrovnalost s jednou z nich, a to Intel Trace collectorem. Tuto nesrovnalost se pokoušíme objasnit pomocí testů. Dále je popsán Metis, což je knihovna numerických metod, pomocí které je vytvářena síť pro výpočet. Následně je tato síť dělena. Další část textu je věnována samotnému testování. Nejprve pomocí knihovny BDDCML, kde jsou rozebrány použité metody i jejich výsledný vliv. Poté je nahlédnuto na vliv těchto změn při použití knihovny Petsc. Tato práce navazuje na můj projekt Vliv MPI prostředí na škálovatelnost paralelních programů.

1 Příprava prostředí Metacentrum

Prvním cílem bylo nalézt vhodné prostředí pro běh flow123d. Toto prostředí by mělo obsahovat všechny potřebné knihovny, podporu mpi aplikací i dostatečný výkon. To vše splňuje Metacentrum, které zajišťuje a koordinuje provoz distribuované výpočetní infrastruktury pod jednotným prostředím. Disponuje také mnoha knihovnami, které může uživatel použít. Vše se ovládá přes SSH (Secure Shell), kdy má uživatel přístup ke stroji pomocí příkazového řádku, zde pomocí unix příkazů. Metacentrum disponuje různými verzemi knihoven, které se zde nazývají moduly a používají se pro vybrání vhodné verze knihovny.

1.1 Překlad Petsc na metacentru

Petsc je nejrozšířenější knihovnou určenou k paralelním numerickým výpočtům. Vývojáři Petsc byli za tuto knihovnu oceněni cenou Computational Science and Engineering v roce 2015. Knihovna je rozsáhlá a obsahuje spoustu různých řešičů, pro nás je důležitý hlavně BiCGStab. Při překladu flow123d je potřeba mít již tuto knihovnu nainstalovanou, proto prvním krokem bylo nakonfigurovat a nainstalovat knihovny PETSc. V tomto případě byla použita verze 3.4.0. Ty se konfigurují pomocí volání configure s potřebnými parametry. Pro zjednodušení jsou tyto parametry ve skriptu `conf.sh`. Ve skriptu bylo potřeba nastavit proměnnou `PETSC_ARCH`, díky které lze mít na jednou počítači nainstalované více verzí knihoven Petsc. Také proměnnou `PETSC_DIR`, kde se nastaví kořenový adresář pro PETSc. Důležitý je také řádek `OPT+=" --with-debugging=0"`, kde se určí, zda budou knihovny přeloženy v debug nastavení či ne. Debug mód zhoršuje rychlost běhu a pro měření je nepoužitelný. Při kompilaci se také využívá pythonovského souboru `configure.py`, který nastaví všechno potřebné a postup kompilace vypisuje do konzole. Více o Petsc: [11]

1.2 Překlad flow123d na metacentru

Po nainstalování Petsc už je možné nainstalovat flow123d. Stažení flow123d proběhlo naklonováním adresáře pomocí gitlabu. Konkrétně příkazem:

```
git clone https://github.com/flow123d/flow123d.git
```

Pro správnou instalaci a běh je důležité mít nahrané i správné moduly, které budou navzájem kompatibilní. Práce s moduly je velmi snadná a všechny dostupné moduly se nachází

v cestě `/software`. Načtené moduly se vypíší při přihlášení do metacentra, či příkazem `module list`. Modul lze jednoduše nahrát pomocí příkazu `module load <název modulu>`. Obdobně lze modul odebrat pomocí příkazu `module unload <název modulu>`. Je-li potřeba změnit moduly načtené při přihlášení, stačí editovat soubor .bashrc. V práci budou využité 2 MPI knihovny, protože intelMPI je potřeba pro použití Intel trace collector a Openmpi byla použita kvůli problému, který se průběhu objevil. Pro správný běh používaných MPI knihoven je potřeba mít nahrané tyto moduly:
Pro Openmpi:
`openmpi, cmake-2.8, gcc-4.8.4, python-2.7.6-gcc`
Pro intelMPI:
`intelcdk-15, boost-1.56-gcc, cmake-2.8, python27-modules-gcc` a `intelmpi-5.0.1`.

Pro překlad flow123d je nutno nejdříve soubor `config.cmake.template` překopírovat do souboru `config.cmake`. V něm nastavit cestu ke knihovně `PETSc` a správně uvést proměnnou `PETSC_ARCH` a `PETSC_DIR` . Samotný překlad probíhá pomocí zadání příkazu `make all`. Ke kompilacije využita rutina `Cmake` více o Cmake zde: [5]. Do proměnné `FLOW_CC_FLAGS` je možné nastavit parametry překladu, například `-g` znamená debug mód, `-DDEBUG_PROFILER` zase zajistí vytvoření `profilel_info souborů`, které obsahují pro měření důležité informace, například čas strávený v jednotlivých částech výpočtu. V novějších verzích flow123d je jejich generování v základním nastavení.

2 Spouštění programů pomocí PBS

Systém dávkového spouštění úloh PBS (Portable Batch System) je určen ke spouštění úloh na výpočetních zdrojích metacentra. Jeho účelem je co nejefektivnější rozdělení mezi dostupné stroje. Pomocí něj také mohou uživatelé specifikovat své požadavky na cílový stroj, běh programu či potřebné moduly. Více zde: [3]. Pro přihlášení se používají čelní uzly (frontendy). Ty pomocí níže uvedeného příkazu `qsub` žádají o čas a prostředky na výpočetních strojích.

Pro zaslání úlohy do fronty slouží příkaz `qsub`. Pro specifikaci požadavků se používají parametry. Nyní uvedu některé parametry, které se používaly v souvislosti se spuštěním úloh pomocí flow123d:

`nodes` - počet strojů, na kterých úloha poběží.

`ppn` – počet procesorů, použitých pro běh na jednom stroji

`mem` – velikost přidělené paměti RAM

`infiniband` – přidělí úloze stroj s vlastností infiniband, která zrychluje běh MPI úloh

`cl_(název stroje)` - spustí úlohu na stroji s daným jménem

Pustí-li se úloha bez parametrů, či se nějaký parametr vynechá, použije se základní nastavení. K zadání parametrů dané úloze se používá přepínač –l, následovaný konkrétními nastaveními oddělenými dvojtečkou.

```
qsub -l nastavení1:nastavení2:...:nastaveníN
```

Chceme-li například spustit úlohu na 1 stroji a čtyřech procesorech, použije se:

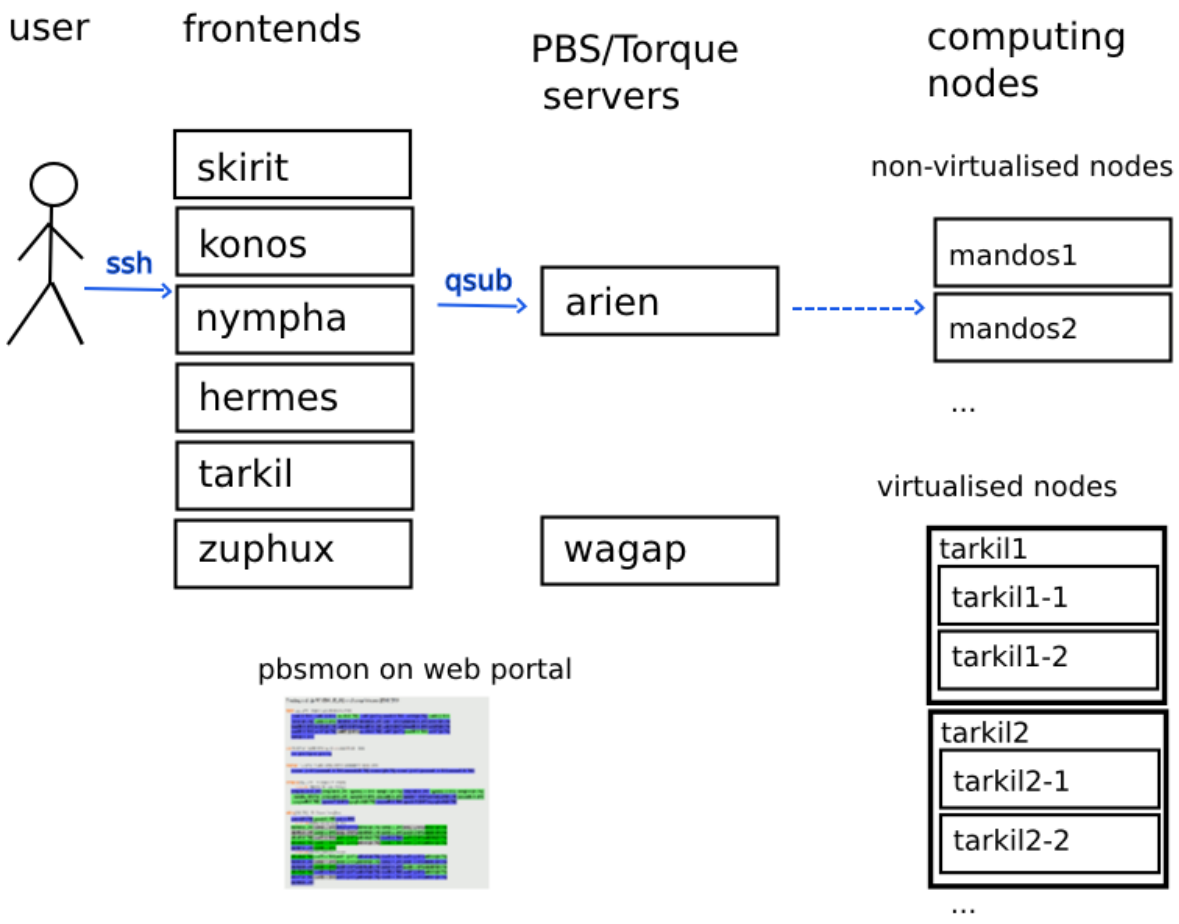
```
qsub -l nodes=1:ppn=4
```

Každá úloha dostane přidělené id, podle kterého ji lze jednoznačně identifikovat. Stav úlohy lze ověřit příkazem

```
qstat -f <číslo úlohy>
```

Úlohy flow123d byly spouštěny dávkově přes qsub. Úloha se v tomto spouští pomocí skriptu flow123d.sh. Zde se zpracovávají některé parametry, které se nacházejí za skriptem. Poté se píší dvě pomlčky („–„) za ně parametry pro flow123d. Důležité je, že samotný qsub příkaz se netvoří zde, ale v souboru `tarkil.cesnet.sh`, proto je nutné zadávat parametry v syntaxi pro flow123d. V tomto skriptu se také načítají moduly, aby na stroji, kde flow123d běží, byly nastavené správné knihovny. Úloha ke spuštění se zařadí do fronty a čeká, až budou požadované prostředky na výpočetním stroji volné. Po jejich uvolnění se spustí výpočet. Standardní výstup z výpočtu úlohy se vygeneruje v podobě souboru `flow123d.o<číslo úlohy>`.Tento soubor se vygeneruje v adresáři, kde byla úloha spuštěna. Příklad spuštění úlohy:

```
~/flow123d/bin/flow123d.sh -- -s ~/uloha1/<nazev>.con
```



Obrázek 1: shrnutí fungování příkazu qsub [3]

Tento obrázek byl zvolen na základě přehlednosti a jednoduchého znázornění děje v Metacentru. Ukazuje cestu úlohy od uživatele, který se nejdříve musí přihlásit na jeden z čelních uzlů (frondend), tam z jednotného prostředí Metacentra zažádá pomocí qsub o výpočetní stroje. Jeho požadavky pracovává PBS/toque server, který drží tyto požadavky, dokud se neuvolní potřebný počet výpočetních uzlů, nebo neskončí nastavená životnost úlohy. Existují zde i virtuální uzly. Což znamená, že existuje jeden fyzický stroj, ze kterého je pomocí softwaru (v Metacenru je využito softwaru s názvem XEN) vyrobeno několik strojů.

3 Měřící nástroje

K zjištění působení změn bylo potřeba měření. Měřili jsme čas zabraný řešičem, počet iterací vyvolané řešičem, čas 1 iterace i čas běhu celého programu. Pro měření času

bylo použito několika nástrojů, k dispozici jsme měli 2 nástroje, a to `log_summary` z Petsc a `profiler_info` z flow123d. K těmto dvěma nástrojům jsme použili Intel trace collector, který poskytuje velmi detailní informace o ději v MPI aplikaci. Například jdou zobrazit všechna MPI volání a poskytuje celkově velmi podrobné informace o efektivitě MPI aplikace. Tyto informace ukládá Intel trace collector do trasovacího souboru s koncovkou .stf. Úloha byla spouštěna pomocí mpirun spouštěče ve verzi intel-mpi, protože Intel trace analyzer vyžaduje tuto knihovnu. Úloha byla vždy spouštěna s parametrem -trace s generováním trasovacích souborů na vzdáleném stroji podporující infiniband. Spouštění probíhalo od 1 do 8 procesorů. Níže jsem popsala všechny použité měřící metody.

3.1 Log summary z Petsc

Tuto volbu poskytuje knihovna Petsc. To aktivuje výpis shrnutí výkonnosti, časy či počty volání v jednotlivých funkcích. Na začátku výpisu ne nachází shrnutí celého běhu. Zde je ukázka: Z této části byla užitečná označená část, a to při porovnání výsledků pomocí několika

```
Summary of Stages:   ----- Time ------  ----- Flops -----  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total   counts   %Total     Avg         %Total   counts   %Total
 0:      Main Stage: 5.1423e+01 100.0%  4.0928e+06 100.0%  4.222e+03 100.0%  1.543e+05      100.0%  1.560e+02  99.4%
```

Obrázek 2: shrnutí z Petsc Log summary

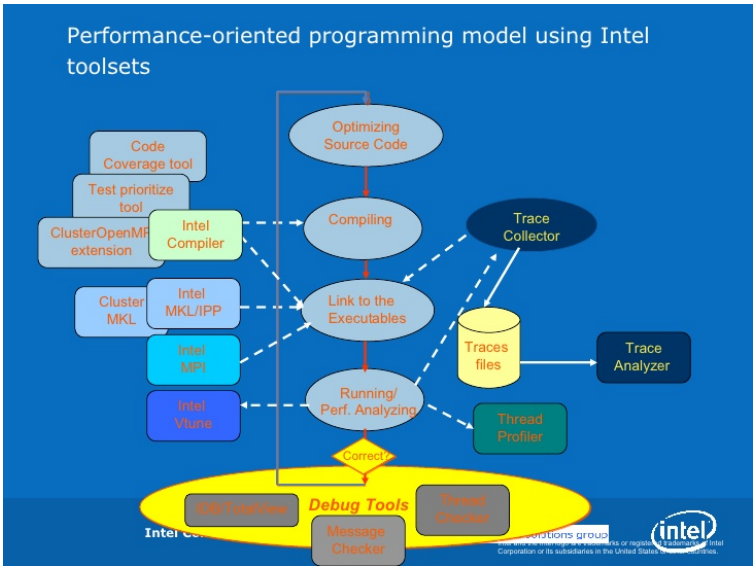
metod. Dále už je podrobný výpis všech metod, informace o čase stráveném vdané metodě, počet všech zpráv poslaných metodou i procento redukčních operací v dané části.

3.2 Profiler info z flow123d

Každý běh úlohy do složky s výstupem vygeneruje mimo jiné i soubor z informacemi o běhu programu. Je vytvořen ve dvou verzích. Jedna jako soubor json, druhá v textové podobě. Informace z něj jsou v něčem podobné jako z log summary, ale ne totožné. V stromové struktuře je zde rozepsáno více metod, počet volání dané metody i funkce v ní obsažené. Stejně také obsahuje časy strávené v jednotlivých metodách a maximální čas strávený v metodě na 1 procesoru.

3.3 Intel Trace collector a Analyzer

Intel trace collector je nástroj na sledování průběhu MPI aplikací. Zaznamenává veškerá mpi volání v programu. Má možnost generovat trasovací soubory, které lze analyzovat pomocí Intel trace collector. Nástroj byl dříve znám jako `Vampirtrace * (VT)` , což je důvodem, proč se zkratka VT nachází v některých jménech komponent a proměnných. Intel trace collector se spouští zadáním příkazu `traceanalyzer` do příkazového řádku. [4] Pro správnou funkčnost musí být na aktuálním stroji nahrán modul s licencí. Po spuštění a výběru trasovacího souboru program zobrazí `Summary Page`, kde jsou informace mimo jiné i o celkovém času běhu programu, kolik času zabraly mpi operace a kolik samotný výpočet. Po tomto shrnutí je další část, kde si již lze podrobněji prohlédnout různé části záznamu a podívat se například, kde na sebe musely procesory čekat nebo kde se nachází mpi volání. Tento nástroj se jevilo jako vhodné použít pro detailnost výsledků, které ostatní metody nenabízejí, ale při používání tohoto prostředku jsme narazili na dva problémy, které budou uvedeny dále v textu.



Obrázek 3: diagram průběhu kompilace

zdroj obrázku [1]

Na obrázku je ukázka ladění kódu pomocí nástrojů Intel Trace collector a Analyzer. Je zde vidět že kompilace kódu probíhá pomocí Intel MPI. Intel trace collector při běhu kompletuje

[1] http://www.slideshare.net/51lecture/no11-185061

data a vytváří trasovací soubory s koncovkou .stf. Trasovací soubory lze zobrazit pomocí aplikace intel trace analyzer.

3.4 Problém při generování trasovacích souborů

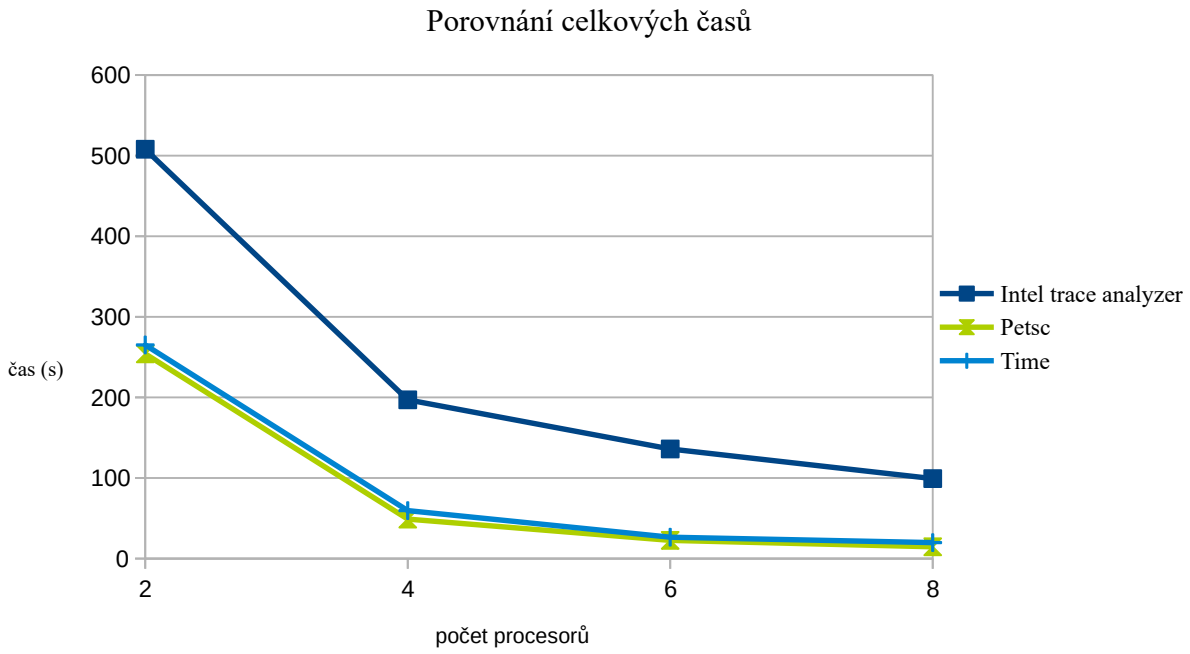
Při vytváření trasovacích souborů se vyskytl problém u spouštění na dvou a čtyřech procesorech, přesněji jejich zápis skončil vždy chybou. Tato chyba byla způsobena pravděpodobně přílišnou velikostí souborů, které nebylo možné zapsat do lokálního adresáře na vzdáleném stroji, kde úloha běžela. Důvodem byl limit velikosti souborů. Jako nejjednodušší řešení je jevilo přesměrovat tento výstup někam, kde tento problém nebude. Jako první řešení byl použit domovský adresář, složka moje_tmp. Přesměrování proběhlo pomocí přiřazení cesty do proměnné VT_FLUSH_PREFIX. konkrétně:

```
export VT_FLUSH_PREFIX=/auto/praha1/lenka_jandurova/moje_tmp
```

Toto řešení bylo úspěšné, bylo ale potřeba zjistit, zda zápis tímto způsobem neovlivňuje i výsledné časy běhu. Při zápisu do mého domovského adresáře je nutná navíc komunikace přes síť, která by mohla výsledné časy ovlivnit. Pro testování bylo použito několikero spuštění na 6 procesorech oběma možnými způsoby. Zbylé nastavení běhu bylo ponecháno stejné, i stroj pro běh byl zvolen totožný. Po porovnání si byly výsledné časy až na zanedbatelné odchylky stejné, proto jsme zvolili tento postup.

3.5 Výsledky měření pomocí několika metod

Výsledky všech metod jsou vždy z jednoho běhu úlohy a na něm byly spuštěny všechny metody měření. Měřeno bylo od dvou po 8 procesorů, s požadováním funkce infiniband. Pro měření byly použity Intel trace collector, log summary z petsc a příkaz time z unixu. Profilovací informace pro shodná data s log summary zde nebyla zahrnuta.



Obrázek 4: porovnání výsledných časů pomocí několika metod

Časy jsou vždy z jednoho běhu, ale měřeny několika metodami. Z výsledků je patrné, že intel trace analyzer má časy velmi větší oproti ostatním metodám měření. Důvod těchto rozdílů vypadá na problém s redukčními operacemi. Proto budou redukční operace otestovány v následující kapitole.

3.6 Měření rychlosti redukčních operací u intel MPI a Open MPI

Pro nevyváženost při použití knihovny intel MPI benchmarků bylo vhodné otestovat i rychlost redukčních operací. K porovnání výsledků byla zvolena knihovna Open MPI a na testování bylo využito mpi benchmarků. Byly použity OSU Micro-Benchmarks 4.4.1 ze zdroje [9]. Test byl prováděn pod oběma knihovnami a při testech byla vždy požadována vlastnost infiniband a jméno konkrétního stroje.

V tabulkách níže je uveden průměr běhu na 1 až 32 procesorech. V tabulce 3.6 chybí

Tabulka 1: Openmpi a Intel MPI naměřené časy

Open MPI					
počet procesorů	2	4	8	16	32
ppn	stroj: minos				
1	6,110	9,800	14,910	20,171	26,323
2	3,780	9,267	14,503	20,486	27,599
ppn	stroj: hildor				
1	3,320	7,034	11,779	16,089	23,026
Intel MPI					
počet procesorů	2	4	8	16	32
ppn	stroj: minos				
1	143,263	290,896	432,499	771,804	–
2	8,767	163,120	328,864	583,406	–
ppn	stroj: hildor				
1	111,893	295,287	482,084	584,016	–

výsledky z Intel MPI pro 32 procesorů, jelikož úlohy pod tímto nastavením nedobíhaly a výsledky z nich nejsou k dispozici. V těchto testech dopadá intel MPI hůře než open MPI. I v rámci knihoven jsou vidět drobné rozdíly, podle daného stroje. Jedině pokud úloha běžela na 1 stroji bez nutnosti komunikace přes síť (viz. ppn 2 na dvou procesorech) výsledky nevypadají tak špatně. Problém bude patrně někde zde a proto bylo rozhodnuto prostředků intel mpi nevyužívat, i přes velké možnosti nabízející Intel trace analyzerem při analýze běhu mpi programů. Použity budou pouze zbylé metody a dále je již využívána jen knihovna Open

MPI.

4 Vliv na dělení sítě na efektivnost výpočtu

Jádrem numerických simulátorů (například flow123d) je řešič soustav lineárních rovnic. V případě flow123d probíhá paralelně. Paralelní řešení soustav vyžaduje distribuci matic a vektorů na jednotlivé procesy. Tato distribuce odpovídá distribucí výpočetní sítě. Dělení sítě je optimalizační úlohou, jejíž cílem je minimalizovat rozhraní mezi díly při zachování stejné velikosti dílů. Jako první byl pro měření použit řešič BDDC. U něj by měl být vliv změn patrný, ale jak dalece, bylo potřeba zjistit. BDDCML (Balancing Domain Decomposition by Constraints - Multi-Level) je knihovna určená k řešení velkých řídkých lineárních soustav s konečným počtem prvků. Umožňuje distribuci výpočtu mezi procesory. [6] K dělení sítě se používá knihovny metis.

4.1 Metis

Síť je dělena pomocí Metisu [8]. Síť je rozdělena na graf obsahující vrcholy a hrany. Tento graf se snaží rozdělit mezi procesory tak, aby každý z nich dostal stejně velkou část grafu a zároveň aby komunikace mezi rozdělenými segmenty byla co nejmenší. Jednotlivým elementům je možné přiřadit i váhu (zde ji budu označovat písmenem q), která má vliv při rozdělování grafu na subdomény. Je možné nastavit váhu q jak hranám, tak i vrcholům grafu. Tato váha má však jiný význam a nebude použita, zmíněna je pouze pro zajímavost. Při generování grafu se vychází s hodnoty seed. Je to náhodná hodnota ale její zadání umožňuje opakovatelnost měření. Zajímavý pohled na Metis a jeho efektivnost a mýty kolem něj popisuje autor ve článku [10].

4.2 Vliv dělení sítě na konvergenci řešiče bddc

Podobné testy byly prováděny například na síti Bedřichov tunel [1]. Tato síť byla složitější a větší než ta, která byla použita ve zdejších testech. Dle původního předpokladu by měl počet iterací řešiče (v tabulce sloupec its.) buď zůstávat konstantní nebo mírně růst. Výsledky ale vykazují místy podivné chování této hodnoty, například na 128 procesorech ve sloupci

aritmetický průměr je tato hodnota výrazně vyšší než by z vývoje odpovídalo. Pro vylepšení bylo využito škálování, ale jak je z posledního sloupce patrné, problém to sice trochu zmenšilo, ale ten stále vliv na výsledky má,(viz počet iterací na 32 procesorech, je zase podezřele vysoký). Důvodem by mohlo být špatné rozdělení vodivějších částí, protože časté střídání vodivosti v jednom elementu může Metisu komplikovat výpočet.

N	n/N	n_Γ	n_f	n_c	aritmetický průměr		mod. ρ-scal.		diagonal scal.	
					its.	cond.	its.	cond.	its.	cond.
32	245k	20k	106	322	637	9811.7	110	1467.8	112	1514.1
64	123k	28k	192	597	618	10254.1	62	115.1	63	117.7
128	61k	45k	413	1293	2834	1.0e+11	206	401641.4	75	194.4
256	31k	72k	902	2791	799	11172.9	117	512.9	119	526.7
512	15k	110k	2009	6347	883	15449.6	136	1160.1	137	1143.4
1024	8k	155k	4575	14725	n/a	2.5e+10	504	99023.6	173	897.0

Tabulka 2: Porovnání různých zprůměrovaných hodnot pro síť bedřichov tunel, obsahující 2D a 3D elementy, velikost globálního problému je $n = 7.8$M neznámých.

4.3 Příprava testovací úlohy

Úloha se skládá ze souboru s příponou con vycházejícího z JSON a ze sítě.

CON soubor řeší proudění (soustava rovnic řešena paralelně metodou BiCGStab z PETSc nebo metodou BDDC z knihovny BDDCML) a transport (lineární soustava je řešena stejně). Soubor s koncovkou msh je mesh síť na které výpočet proudění probíhá.

Cílem bylo vytvořit dostatečně velkou síť, aby výpočet zabral větší množství času. K vytvoření msh byl použit program gmsh [2], pomocí kterého byla upravena stávající úloha 1 a zvětšena na 240 000 elementů. Tato velikost síte se pro danou úlohu jevila jako nejvhodnější. U větších sítí byl pro tuto úlohu problém s velikostí elementů, byly příliš malé. Navíc byl zaznamenám problém s pamětí při výpočtu velkých úloh.

Zaznamenali jsme problém s testovací úlohou, které nestačilo přidělených 400Mib RAM udělených v původním nastavení. Pro vyřešení pomohlo v qsub příkazu žádat o 4Gb paměti. Příklad ze nastavení požadavku na 4GB paměti v skriptu pro generovaní qsub:

```
PBS -l mem=4gb
```

a díky tomu se všechny qsub úlohy generovaly s požadavkem na 4 GB paměti.

4.4 Vlastní testování sítě

Úlohy byly spouštěny s řešičem bddc. Spouštět úlohu na 1 procesoru řešič bddc neumožňuje. Při spouštění testovací úlohy s bddc řešičem se u nižších počtů procesorů objevil problém pádu aplikace na chybě `segmentation fault` a vygenerování souborů core. Tato chyba se jevila jako problém s nedostatkem paměti aplikace při běhu. Nakonec se nám podařilo zjistit, že chybou je přetečení zásobníku (stack overflow) a to i vysvětlení, proč zvětšení paměti nepomohlo. Nyní stačí před spuštěním zažádat o větší paměť zásobníku a výpočty probíhají v pořádku.

4.5 Změna váhy elementů na puklině

V programu flow123d jsme implementovali změnu váhy elementů. Zvolili jsme elementy nacházející se na puklině. Cílem bylo, aby puklina byla rozdělena na co nejméně subdomén, a zároveň, aby se čas strávený nad jednou iterací nezvětšil. V souboru mesh/partitiong.cc při vytváření grafu, ve funkci `graph_->set_edge()` pro vytvoření hran grafu, jsme nasta-

vili jednotlivým hranám i váhu q. Jelikož je potřeba hodnotu určující váhu vícekrát měnit, bylo potřeba ji zadávat zvenčí, bez opětné kompilace kódu. Pro byla použita proměnná `2d_weight`, kterou jsme zavedli v kódu programu. Hodnota této proměnné se naplňuje v konfiguračním souboru .con v části mesh a v ní vnořené partitioning:

```
  mesh = {
mesh_file = "./input/cube_2D_diagonal.msh",


partitioning = {
"2d_weight" = 2
}
},
```

4.5.1 Skript pro sběr dat

Pro sběr většího množství výsledků bylo ruční kopírování příliš pomalé řešení. Proto jsme použili rychlejšího sběru dat pomocí skriptu. Ten je napsán v programovacím jazyku python, pro snadné zpracování json souboru. Python obsahuje knihovnu, která umožňuje snadnou prácí s tímto typem souboru. Skript nejprve prohledá všechny podadresáře a vyhledá všechny json soubory. V souboru jsou data ve stromové struktuře, uložena v potomcích. Z těch pomocí samostatné funkce vyhledává údaje, které potřebujeme a ukládají se do výstupního souboru. U špatně formátovaného vstupu pouze informuje o jeho neplatnosti. Skript je psán s ohledem, že byl využíván jak pod operační systémem Linux, tak i Windows.

4.5.2 Testování a výsledky měření

Pro testování byly zvoleny váhy 1,3,10,30,100,300,1000. Z výsledků běhů s různými vahami byly podstatné tyto informace:

- délka běhu celého programu,
- počet iterací z profiler_info(řádek BDDC linear iteration)
- čas jedné iterace
- `DarcyFlowMHOutput::make_node_scalar_param`

- Rozdělení plochy pukliny pro jednotlivé subdomény

Délka běhu celého programu byla zvolena jako orientační údaj, na počtu iterací je dobře vidět účinek změn a pomocí něho se počítá i čas 1 iterace.
`DarcyFlowMHOutput::make_node_scalar_param` byl použit na redukci časů běhů ovlivněných strojem. Tyhle data jsme získali z profilovacích informací. Rozdělení plochy pukliny pro jednotlivé subdomény je z programu paraview. Dále bylo potřeba eliminovat nepřesnosti v měřených časech způsobené různou výpočetní silou strojů s vlastností infiniband. Možnosti byly dvě. Buď vybrat funkci závislou pouze na stroji, nikoli na počtu procesorů. Hodnotou této funkce poté časy podělit. Nebo při spouštění úloh vyžadovat stále stejný stroj. Původně se jevila jako snazší možnost se stále stejným strojem. V průběhu měření se tato možnost ukázala jako nepoužitelná pro pomalost výsledků a v některých případech i nedodání výsledků vůbec. Spuštění na náhodném stroji, sice s vlastností infiniband, ovlivnilo časy měření o výpočetní sílu stroje a aktuální stav v metacentru. Proto bylo potřeba nalézt parametr, který výsledky alespoň částečně oprostí od vlivu stroje. Jako vhodný kandidát se jevil `DarcyFlowMHOutput::make_node_scalar_param`. Před jeho použitím bylo potřeba otestovat, zda pro stejný stroj, ale různý počet procesorů hodnota odpovídá. V následující tabulce jsou hodnoty pro tři stroje, kde úloha proběhla jak na 16, tak 32 procesorech.

Tabulka 3: porovnání strojů

název stroje	**32 procesorů**	**16 procesorů**
hildor	2.0335	2.08890
mandos	4.206	3.7068
alfrid	2.3086	2.58

Výsledky nejsou úplně totožné, ale podobnost je zřejmá, proto jsou všechny následující časy děleny touto hodnotou.

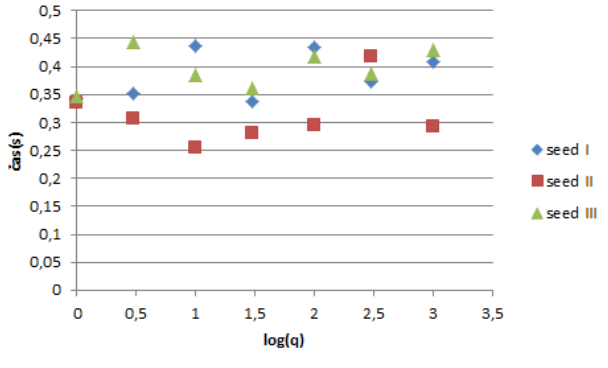
Nyní se podíváme blíže na výsledky běhů na 32 a 16 procesorech. Data jsou z 5ti běhů a 3 hodnot seed. Z výsledků vyplynula zajímavá věc, a to, že počty iterací jsou i při použití stejné váhy a hodnoty seed na identickém počtu místy různé, což by se dít nemělo. V případech, kdy se výsledky pro daný seed drobně liší, je použita nejčastěji se opakující naměřená hodnota pro daný seed. Použité časy jsou aritmetickým průměrem za všech běhů.

Tabulka 4: výsledky měření z 32 a 16 procesorů

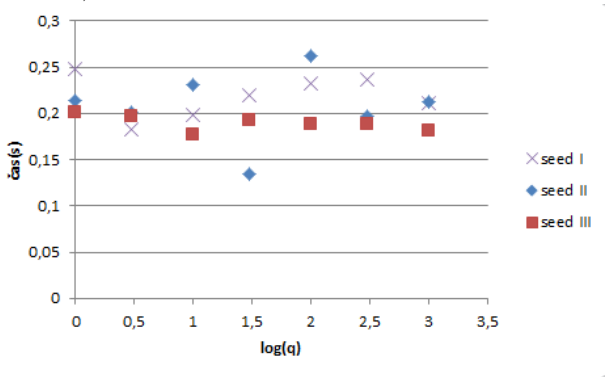
32 procesorů								
iterace počet	log(q)	0	0.477	1	1.477	2	2.477	3
	Seed I	41	41	34	35	30	30	33
	Seed II	39	38	36	31	33	31	36
	Seed III	36	37	31	36	25	31	32
setup čas (s)		4,057	3,796	3,891	3,805	4,16	4,202	4,202
1 iterace čas (s)		0,221	0,194	0,202	0,182	0,228	0,207	0,207
16 procesorů								
iterace počet	log(q)	0	0.477	1	1.477	2	2.477	3
	Seed I	31	23	28	24	24	29	25
	Seed II	32	23	22	21	24	24	24
	Seed III	30	26	27	24	24	27	23
setup čas (s)		8,852	9,724	9,213	9,183	9,655	10,488	9,77
1 iterace čas (s)		0,342	0,368	0,359	0,326	0,383	0,393	0,377

Grafy 4.5.2 obsahují čas 1 iterace z 16 a 32 procesorů a různých hodnot seed. I po normalizaci časů v závislosti na stroji jsou znát vlivy z aktuálního stavu v metacentru a obsahují mírné zašumění. Tyto grafy ale ukazují, že čas na hodnotě seed nezávisí.

Obrázek 5: 16 a 32 procesorů, čas 1 iterace



Obrázek 6: 16 procesorů - čas 1 iterace



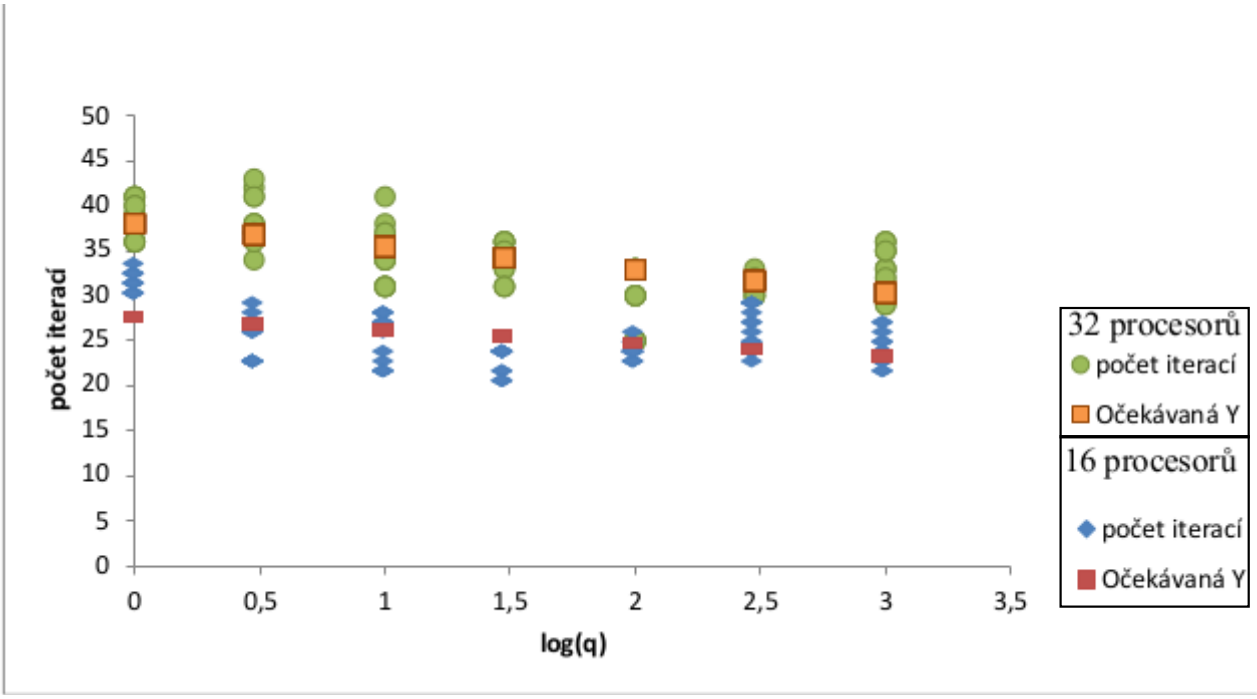
Obrázek 7: 32 procesorů - čas 1 iterace

Pro lepší přehled o vývoji dat jsme pro všechna data z jednoho měření, tedy počet iterací a čas jedné iterace, spočítali regresi a vykreslili graf porovnání hodnot. V následujících tabulkách jsou výsledky. Regrese byla počítána ze všech dostupných dat v programu Excel, který také vykreslil graf regresní přímky. V tabulce s regresí koeficient u závislosti na váze q ukazuje vývoj hodnot. V tomto případě je hodnota záporná. To znamená, že hodnota sledované veličiny se v závislosti na váze q snižuje. Nulová hodnota by znamenala, že vliv není žádný a kladná hodnota, že by sledovaná veličina rostla. Důležitá je také hodnota P, která ukazuje, s jakou pravděpodobností je tento ukazatel nenulový.

Tabulka 5: 16 procesorů,počet iterací

16 procesorů	Koeficienty	Chyba stř. hodnoty	t Stat	Hodnota P
Průměrný počet iterací	27,508	0,472	58,333	1,01E-80
Závislost na váze q	-1,329	0,263	-5,060	-1,84E-06
32 procesorů	Koeficienty	Chyba stř. hodnoty	t Stat	Hodnota P
průměrný počet iterací	37,986	0,538	70,576	4,90316E-89
závislost na váze q	-2,560	0,300	-8,535	1,30839E-13

Další obrázek ukazuje vývoj hodnot pomocí regresní přímky. Modré a zelené hodnoty jsou naměřené hodnoty, pod očekávané Y jsou hodnoty jsou body z regresní přímky. Zde je vidět mírná sestupná tendence počtu iterací jak u 16, tak 32 procesorů.



Obrázek 8: 16 a 32 procesorů, regrese, počet iterací, souhrnná tabulka

Stejné výpočty jsme provedli i pro čas 1 iterace. Zde bylo potřeba zjistit, jaký má vliv snižující se počet iterací na čas 1 iterace.

Tabulka 6: 16 procesorů,čas 1 iterace

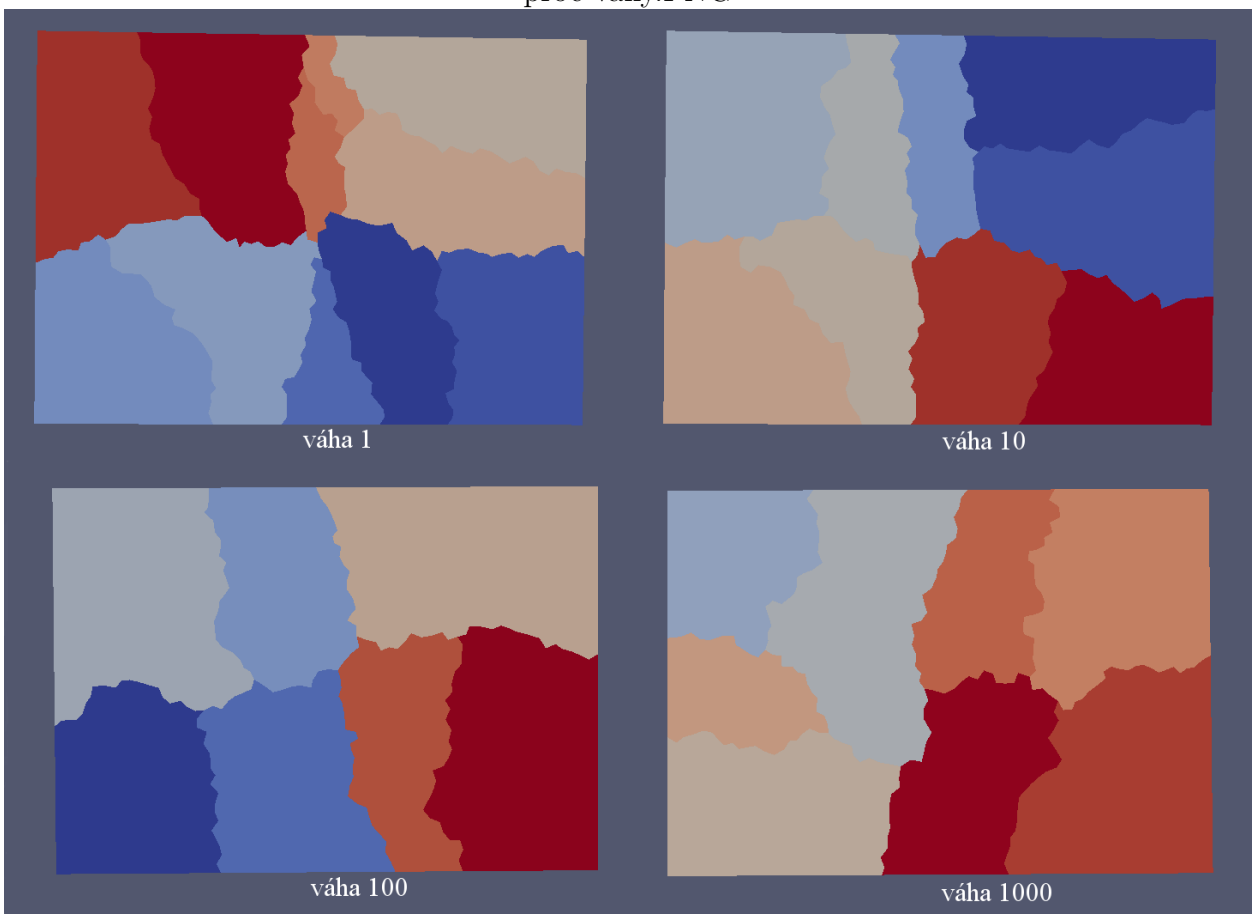
16 procesorů	Koeficienty	Chyba stř. hodnoty	t Stat	Hodnota P
Průměrný čas	0,344	0,029	11,957	3,46254E-21
závislost na váze q	0,013	0,016	0,809	0,420130408
32 procesorů	Koeficienty	Chyba stř. hodnoty	t Stat	Hodnota P
Průměrný čas	0,205	0,015	13,753	4,67038E-25
závislot na váze q	0,000	0,008	-0,018	0,985369036

Z regrese vyplývá, že na čas 1 iterace nemá rostoucí váha vliv, jak ukazují koeficienty v závislosti na váze q, které se blíží 0. Což je dobře, že i přes snižující se počet iterací řešiče se nezvyšuje čas 1 iterace.

4.6 Změna hodnoty seed při řešení proudění

Cílem bylo zjistit, jak rozložení procesorů na puklině ovlivní změna této hodnoty. Hodnotu seed je možné změnit v souboru `la/sparse_graph.cc`, v části `SETTING OPTIONS`. V poli options lze nastavit klíč `METIS_OPTION_SEED` na hodnotu, kterou bude potřeba. K testování byly použity náhodně zvolené hodnoty. Pro změnu této hodnoty byl do kódu přidán klíč seed, který se také nastavuje v konfiguračním souboru úlohy, aby nebylo nutné stále kompilovat kód. Nastavit lze také, jak METIS [8] tvoří graf z elementů. Pro větší náhodnost při vytváření rozdělení sítě bylo přiděleno do klíče `IPTYPE` hodnotu `METIS_IPTYPE_RANDOM`. Cílem je dosáhnout nižšího počtu subdomén, narovnat rozhraní subdomén na puklině a eliminovat velmi malé subdomény z pukliny.

proc-vahy.PNG



Obrázek 9: ukázka několika rozvržení subdomén na puklině

Obrázky obsahují grafické znázornění rozložení subdomén na puklině, části sítě s vyšší vodivostí. Měření probíhalo na 16cti procesorech, na náhodně vygenerované hodnotě seed a na popsaných vahách. Z obrázků je patrné, že váha má vliv na rozložení subdomén na puklině i na jejich počet, který mírně klesá. Na obrázcích je vidět i viditelné narovnání subdomén na puklině. Počet subdomén k jednotlivým vahám a z více měření je pro větší přehlednost i níže v tabulce : 16 procesorů porovnání počtu subdomén.

Tabulka 7: 16 procesorů porovnání počtu subdomén

váha	1	3	10	30	100	300	1000
počet subdomén	11	11	9	7	8	8	8
počet subdomén	12	9	8	8	7	7	8

Z tabulky je patrné, že použitá váha má na počet subdomén vliv. Klesá jen do váhy 300, dále se drží na stejné hodnotě. Rozložení pukliny mezi jednotlivými procesory je pro více běhů se stejnou vahou i počet procesorů totožné.

4.7 Vliv změn na řešič BiCGStab z Petsc

Jako druhý řešič pro testování vlivu byl zvolen řešič Petsc. Pro změnu stačí v souboru úlohy změnit část `Solver` hodnotu `TYPE` na Petsc. Úloha byla spouštěna s požadavkem na funkci infinibend a bez požadavku na konkrétní stroj. Výsledky jsou ze souboru profiler info, ze kterého jsou hodnoty počtu iterací a PETSC linear solver. Tyto hodnoty jsme získávali pomocí upravené verze skriptu pro sběr dat. Hodnota 1 iterace je vypočtena jako `PETSC linear solver/pocet iteraci`. Z výsledků bylo na první pohled patrné, že ani změna hodnoty seed, ani váhy nemají na výsledné časy či počty iterací viditelný vliv. Patrný je pouze rozdíl v čase dle výpočetní síly stroje. Zde jsem také spočítala regresi, pro lepší představu o vlivu váhy q.

Tabulka 8: 16 proc petsc

Počet iterací				
	Koeficient	Chyba stř. hodnoty	t Stat	Hodnota P
Průměr	51,2905	0,5948	86,2331	1,8942E-65
Závislost na váze q	0,1565	0,3314	0,4723	0,6384
Čas jedné iterace				
	Koeficient	Chyba stř. hodnoty	t Stat	Hodnota P
Průměr	0,0155	0,0016	9,7324	4,9799E-14
Závislost na váze	0,00056	0,00089	0,6321	0,5297
Petsc solver				
	Koeficient	Chyba stř. hodnoty	t Stat	Hodnota P
Průměr	0,7931	0,0819	9,6887	5,8934E-14
Závislost na váze	0,0315	0,0456	0,6909	0,4923

Závěr z regrese je takový, že není závislost na váze q, zejména nedošlo k zhoršení s výsledků s rostoucí váhou q.

5 Závěr

Prvním závěrem práce je zjištění, že Intel trace collector a Intel MPI jsou pro měření nepoužitelné. Důvodem je patrně nějaký problém v Intel MPI, nikoliv v Intel trace collectoru. Toto tvrzení vyplývá z výsledků MPI benchmarků, kde výsledky redukčních operací v Intel MPI dopadají v průměru několikrát hůře než z Open MPI. Jedinou vyjímkou byly případy, kdy celý výpočet běžel na 1 stroji a nepoužíval síťovou komunikaci. To pak měla knihovna Intel MPI srovnatelné výsledky s Open MPI. Z výsledků testů s knihovnou bddc je zřejmé, že změny v použité váze q mají vliv na počet subdomén a jejich rozložení na puklině. Vliv je pozorovatelný i u počtu iterací řešiče BDDC, jejich počet mírně klesá s rostoucí váhou. Čas jedné iterace viditelným způsobem neroste, což je také správně. U řešiče Petsc není vliv patrný, ale možná by se vliv váhy q mohl projevit při mnohem větších hodnotách váhy q.

Do budoucna by bylo vhodné otestovat dělení na mnohem větších sítích i na větších počtech procesorů, zde jsme byli v těchto testech limitováni možnostmi Metacentra. Dále by se také mělo při testech změřit délky okrajů subdomén na puklině a vyzkoušet použít i větší hodnoty vah q.

Literatura

[1] J. Šístek, J. Březina, B. Sousedík. "BDDC for Mixed-Hybrid Formulation of Flow in Porous Media with Combined Mesh Dimensions." Numerical Linear Algebra with Applications 22, no. 6 (December 1, 2015): 903–29. doi:10.1002/nla.1991.

[2] Gmsh[online]. [cit. 2015-2-28]. Dostupné z: http://geuz.org/gmsh/

[3] Plánovací systém[online]. [cit. 2016-3-30]. Dostupné z: https://wiki.metacentrum.cz/wiki/Pl%C3%A1novac%C3%AD_syst%C3%A9m_-_detailn%C3%AD_popis

[4] Intel trace analyzer[online]. [cit. 2016-3-30] https://software.intel.com/en-us/intel-trace-analyzer/documentation

[5] Ken Martin and Bill Hoffman. Mastering CMake. 6th ed. Clifton Park, NY: Kitware, 2013. ISBN 1930934262.

[6] BDDC[online]. [cit. 2016-4-10] http://users.math.cas.cz/~sistek/software/bddcml/manual_bddcml-2.5.pdf

[7] Parawiew[online]. [cit. 2016-4-11] http://www.paraview.org/

[8] METIS[online]. [cit. 2016-4-11] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[9] OSU Micro-Benchmarks 4.4.1[online]. [cit. 2016-5-14] http://mvapich.cse.ohio-state.edu/benchmarks/

[10] Load balancing fictions, falsehoods and fallacies, Hendrickson, Bruce, Sandinal National Laboratories in the National Interest[online]. [cit. 2016-5-14] http://www.sciencedirect.com/science/article/pii/S0307904X00000421

[11] Petsc[online]. [cit. 2016-5-14] https://www.mcs.anl.gov/petsc/documentation/index.html